第４章

初歩からのHDLテストベンチ

テストベンチの書き方を身に付ける

安岡貴志

本稿ではVerilog HDLとVHDL両方のテストベンチの書き方を解説します．Verilog HDLまたはVHDLのコードを書けるようになったばかりで，これからテストベンチを書こうとしてる方を対象としています．ひとりも脱落者を出さないために，しばしば“くどい”表現や同じ内容のくり返しをすることがありますが，ご容赦ください．（筆者）

あなたがVerilog HDLやVHDL（以降，両方を指すときには単にHDLと言う）で回路を書いたとき，必ずそのコード（HDLの文法で書かれた文章やプログラム）が正しく書けているかを確認しなくてはなりません．なぜなら，そのコードには勘違いや書き間違いをして，思ったものと違う回路になってしまっている可能性があるからです．

ミスは人間であれば仕方がないことです．どんなに経験が豊かであろうと，どんなに注意深く書こうと，ミスというものはなくなりません．設計する回路が複雑で，ソース・コードが長くなり，いくつものファイルに分かれるようになれば，ミスが入り込む可能性は上がります．

1　検証の重要性とテストベンチ

テストベンチ（testbench）とは，HDLで書いた回路が，意図したとおりにできているかどうかを確認（検証；verification）するためのものです．検証では，論理シミュレータをはじめとする回路の動作を模擬するツールを使います．このとき，設計した回路とテストベンチの二つが必要になります．

● テストベンチはHDLで記述する

テストベンチは，回路と同じようにHDLで記述します．

回路の記述ではHDLの文法のすべてが使えるわけではありませんでした．回路を書くときには，物理的な回路として実現できる範囲でしかHDLの文法を使えませんでした．

これに対してテストベンチでは，HDLの文法のすべてを，それこそC言語のプログラムのように使うことができます．逆に言えば，回路を書いている間には使わなかった文法がテストベンチを書く際には必要になってきます．

本稿では，テストベンチの基礎からテストベンチ特有の文法までを，順を追って解説していきます．

● FPGA向けの回路でも検証が必要

FPGA（field programmable gate array）向けの回路の開発でHDLを使う場合，あなたはテストベンチを作ってシミュレーションしなくても，FPGAに搭載して検証すればいいじゃないか，と思うかもしれません．

しかし，期待通りに動かなかった場合には，とたんに効率が下がってしまいます．FPGAのピンをオシロスコープやロジック・アナライザなどで観測しながら，解析しなければならないためです．FPGAの入出力信号だけで不具合の判断ができないときには，内部信号を空きピンに出力する記述を加えなければなりません．観測したい信号の方が空きピンよりも多ければ，ピンに接続する信号を切り替えながら観察するなど，大変な労力が必要になります．例えば，ある内部信号について，1クロック分のパルスが17個でているか確認しようとするだけでも非常に大変です．

KeyWord　Verilog HDL，VHDL，テストベンチ，モジュール，エンティティ宣言，アーキテクチャ宣言，コンフィグレーション宣言，インスタンス宣言，コンポーネント宣言，ライブラリ宣言，パッケージ呼び出し，信号宣言，initial文

これに対し，テストベンチを作ってシミュレーションし，結果を波形で観測する方法を使えば，任意の内部信号を必要なだけ並べて確認することができます．FPGAの開発であっても，最初にシミュレータによって回路の動きを確認し，動作を確実にしてから実機で検証する方が，圧倒的に効率が良くなります．

また，近年FPGAは回路規模が上がり，動作が複雑化しています．すべてが出来上がってから，実機だけで解析することは不可能になっています．

FPGAの開発であっても，テストベンチを作成してシミュレータにより検証することは，もはや必須です．

● 機能ブロック単位で検証する

テストベンチを作るためには，回路を作るのと同じか，場合によってはそれ以上の時間がかかります．ゲートやカウンタ，セレクタなどを一つずつ検証していては，いくら時間があっても終わりません．従って検証は「××処理部」や，「○○制御部」といった機能ブロック単位で行います（**図1**）．

そして，機能ブロックごとの検証が終わってから，検証済みの機能ブロックを接続して，全体の検証を行うようにします．

● 検証の心構え

検証を行うということは，検証に合格した回路が確実に仕様を満たすことを保証する意味を持ちます．

もし，検証に抜けがあって，回路の不具合を見過ごしてしまった場合，あとでそれが発覚して作業をやり直さなければならなくなったり，その開発にかかわるほかの人に迷惑をかけたり，製品に不具合が残れば，最悪製品の回収ということもあり得ます．

そこで，水も漏らさぬ検証を目指しましょう．

2 検証仕様と検証プランを作成する

テストベンチを作る前には，どのように検証するか考えなくてはなりません．そして，検証の内容を考えるためには，検証の対象がどのような機能を実現しなければならないかを知らなくてはなりません．

長々とテストベンチ作成の方法論を語るよりも，例を参考にした方が理解しやすいと思うので，ここからは例を挙げて解説します．

● 検証仕様＝何を検証するのかを考える

ここでは，解説の要点を絞るために，**図2**のような非常に簡単な回路and_combの検証を考えます．

and_combがこの回路の名前です．Verilog HDLであればモジュール名，VHDLであればエンティティ名がそれに当たります．

and_combは，記憶素子（フリップフロップなど）を含まない組み合わせ回路です．1ビットの入力ポートA，Bと1ビットの出力ポートYの間には，**図2（b）**のような関係が成り立つものとします．組み合わせ回路なので，入力ポートに接続されている信号の値が変化すると，出力ポートから出力される信号の値は，直ちに変わります．

図2（b）は，「真理値表」と呼ばれるものです．A列の値は入力ポートAに接続された信号の状態，B列の値は入力ポートBに接続された信号の状態を示し，そのときの出力ポートYの状態がY列の値です．例えば，丸枠で囲まれた行は，ポートA，Bへの入力信号がいずれも‘0’のとき，ポートYからの出力信号は‘0’になるというように読みます．

今回の回路では，入力ポートA，Bに**図2（b）**の四つの

図1　機能ブロック単位で検証する
検証は「××処理部」や「○○制御部」といった機能ブロック単位で行う．機能ブロックごとの検証が終わってから，検証済みの機能ブロックを接続して，全体の検証を行う．

A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

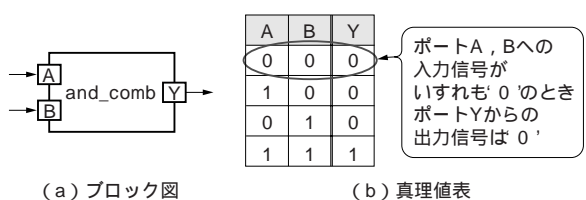



図2　検証対象の回路
回路の名前は，and_combである．1ビットの入力ポートA，Bと1ビットの出力ポートYを持つ．記憶素子（フリップフロップなど）を含まない組み合わせ回路である．

組み合わせを入力し，ポートYからの出力信号が表の通りになっているかを確認する必要があります．この回路にはそれ以外の機能はないので，その四つの組み合わせを評価すれば，十分であるといえます．

お気付きかもしれませんが，これはANDゲートの仕様です．実際には，これほど小規模の回路を検証をすることはありません．しかし，簡単な機能のほうが検証の全容が理解しやすいので，ここではあえて非常に小規模な回路（ゲートがたったの一つ）で，テストベンチの作り方を追って行きたいと思います．

● 検証プラン＝どのように検証するかを考える

検証の内容が決まったら，次にそれをどのような波形で検証対象の回路に入力するかを考えます．

図2(**b**)のAとBの四つの状態の組み合わせを上から順番に入力ポートA，Bに与えるような，入力信号SA，SBと出力信号SYの動作を考えると，**図3**のようになります．ここで信号SAは`and_comb`の入力ポートAに，信号SBはポートBに，信号SYは出力ポートYに接続しているものとします．

図3のタイミング・チャートにおいて，0nsから100nsまでが信号SA，SBが共に‘0’の場合，100nsから200nsまでが信号SAが‘1’で，信号SBが‘0’の場合を表しています．ここでは信号の変化の周期を100nsにしていますが，信号変化の周期は10nsであっても1000nsであっても構いません．また，`and_comb`は組み合わせ回路なので，信号SA，SBが共に‘1’から始まるなど変化の順番に入れ替わりがあっても構いません．大事なのは，ポートA，Bに，**図2**(**b**)の四つの組み合わせすべてが，与えられているかどうかです．

一般的な言い方ではないかもしれませんが，検証のために検証対象の回路に与える信号や値を，本章では「テスト入力」と呼びます．今回は，信号SA，SBがテスト入力になります．

また，**図3**の信号SYの変化は，`and_comb`からの出力信号の「期待値」であって，テスト入力として`and_comb`の外から与える信号ではありません．期待値とは，回路の出力がこうであれば，その回路が求める機能を満たしているといえるものです．

今回であれば，テスト入力として信号SA，SBを**図3**のように`and_comb`の入力ポートA，Bに与えたとき，`and_comb`からの出力として，出力ポートYから**図3**の信号SYのような信号が出力されれば合格です．逆に，出力信号が**図3**の信号SYのようにならなければ，回路を修正しなければなりません．

3　テストベンチを記述する

入力すべき信号の波形が決まったら，いよいよテストベンチの回路（コード）を記述します．一番簡単なテストベンチの形は，**図4**のようになります．

このテストベンチを作る手順を書き並べると，以下のようになります．

1）検証環境を置く箱を作る
2）箱に検証対象の回路を置く
3）箱の中で入力波形を作る
4）信号をテスト対象の回路のポートにつなげる

図3　and_combのタイミング・チャート
図2(b)の真理値表で示されるAとBの四つの状態の組み合わせを，上から順番に入力ポートA，Bに与えるような入力信号SA，SBと出力信号SYの動作を示す．信号SAは`and_comb`の入力ポートAに，信号SBはポートBに，信号SYは出力ポートYに接続しているものとする．

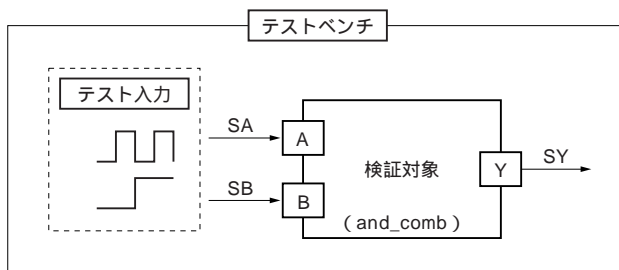



図4　テストベンチの構造
テストベンチの中には，検証対象の回路とテスト入力を生成するブロック，検証対象のポートに接続される信号がある．

● 検証環境を置く箱を作る

ここでいう箱とは，**図4**の一番外側の枠に相当します．HDLによる回路の記述では，Verilog HDLならばモジュール，VHDLならばエンティティという箱を作ったはずです．テストベンチも同じように，モジュールもしくはエンティティという箱を作ることになります．ただし，回路記述（回路の構成や機能をHDLで表現したコード）と違って，この箱には入出力のポートはありません．テストベンチは，シミュレーションを行う際の最上位階層（一番外側の箱）に当たるので，この箱から外に信号を出し入れする必要がないからです．

この箱に`and_comb_tb`という名前を付けることにします．また，`and_comb_tb`を記述するファイルを，Verilog HDLであれば`and_comb_tb.v`，VHDLであれば`and_comb_tb.vhd`とします．

Verilog HDL

リスト1(a)は，Verilog HDLによるモジュールの書式です．

点線の枠内は，回路記述ではモジュール内で使う信号の宣言や回路の機能を記述しました．テストベンチ`and_comb_tb`もこの形式に従って書くことになります．

リスト1(b)は，`and_comb_tb`をVerilog HDLで記述したものです．

回路記述のときには，かっこの中に入出力ポートを記述していましたが，テストベンチではなくなっています．点線の枠内には，これから検証対象となる回路や，テスト入力を生成する記述を埋めていきます．

VHDL

リスト2(a)はVHDLの書式です．VHDLでは一つの箱がエンティティ宣言，アーキテクチャ宣言，コンフィグレーション宣言の三つからなります．

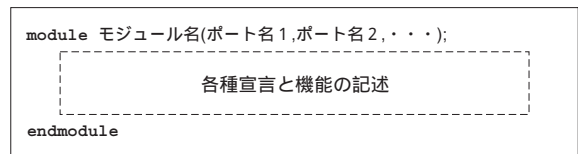
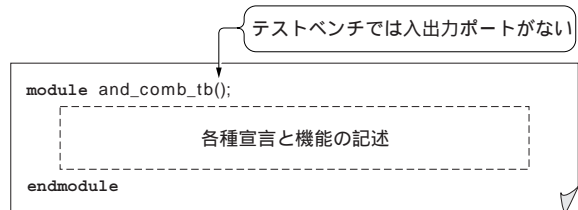
エンティティ宣言は，その箱を外から見てどう見えるかを表したものです．箱の名前と入出力ポートなどを記述します．

アーキテクチャ宣言は，その箱の中がどうなっているか

リスト1　Verilog HDLによるモジュールの書式と記述例

```
module モジュール名(ポート名1,ポート名2,・・・);
    [各種宣言と機能の記述]
endmodule
```

(a) 書式

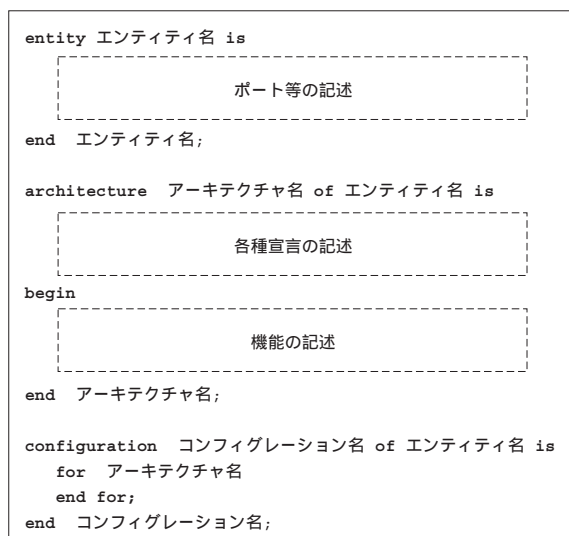
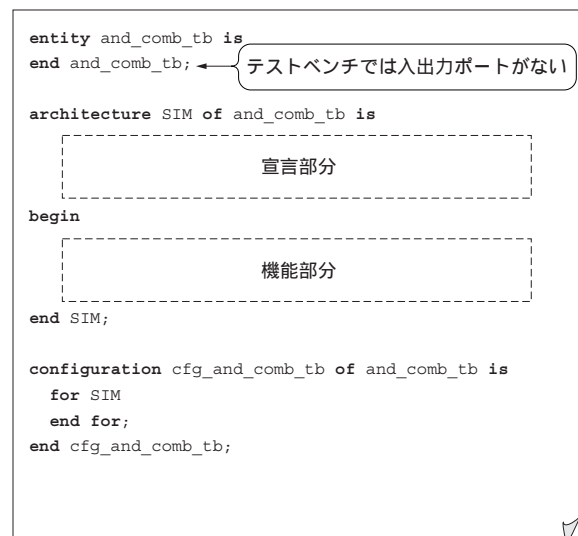
```
module and_comb_tb();
    [各種宣言と機能の記述]
endmodule
```


(b) 記述例

リスト2　VHDLによるエンティティ宣言，アーキテクチャ宣言，コンフィグレーション宣言の書式と記述例

```
entity エンティティ名 is
    [ポート等の記述]
end  エンティティ名;

architecture  アーキテクチャ名 of エンティティ名 is
    [各種宣言の記述]
begin
    [機能の記述]
end  アーキテクチャ名;

configuration  コンフィグレーション名 of エンティティ名 is
   for  アーキテクチャ名
   end for;
end  コンフィグレーション名;
```

(a) 書式

```
entity and_comb_tb is
end and_comb_tb;

architecture SIM of and_comb_tb is
    [宣言部分]
begin
    [機能部分]
end SIM;

configuration cfg_and_comb_tb of and_comb_tb is
  for SIM
  end for;
end cfg_and_comb_tb;
```


(b) 記述例

を表したものです．配置された信号やその接続関係などを記述します．

コンフィグレーション宣言部分では，下層の回路の構成を設定できますが，ここでは何もしない書式になっています．コンフィグレーション宣言は，回路記述では省略も可能なので，VHDL記述の経験が浅い方にとっては初めて見る記述かもしれません．シミュレータによっては，最上位階層にコンフィグレーション宣言がないと動かないものもあります．よって，コンフィグレーション宣言はテストベンチには必ず記述します．

リスト2(b)は，and_comb_tbの箱の記述例です．アーキテクチャ名はSIM，コンフィグレーション名はcfg_and_comb_tbとしています．

アーキテクチャ名は基本的に何でも構いません．VHDLの解説書などの例をみると，回路のときはRTL，テストベンチではSIMとしているものが多いようです．コンフィグレーション名も基本的には何でもよいのですが，ここでは“cfg_＋エンティティ名”としています．

回路記述のときには，エンティティ宣言の中に入出力ポートを記述しましたが，テストベンチではなくなっています．アーキテクチャ宣言の点線の枠内には，これから検証対象となる回路や，テスト入力を生成する記述を埋めていきます．

● 箱に検証対象の回路を置く

Verilog HDL

リスト3(a)はVerilog HDLのインスタンス宣言です．

Verilog HDLでは，回路（仮に親回路とする）の中に別の回路（仮に子回路とする）を置くときに，インスタンス宣言を行います．インスタンス名は親回路の中などで子回路を指定するときの呼び名のようなものです．モジュール名とインスタンス名の関係を，“シバ”という名前の“柴犬”に例えると，モジュール名が“柴犬”でインスタンス名が“シバ”に当たります．インスタンス名があることで，親回路の中に複数の同じ子回路が存在しても，区別することができます．家に柴犬が2匹いるとき，“柴犬”といってもどちらを指すのか分かりませんが，“シバ”と“ソバ”ならどちらか分かるようなものです．

インスタンス名の右のかっこ内に，“.ポート名1（信号名1）”という記述があります．これは子回路のポート1を親回路の信号1に接続することを意味します．ポート名の前に“.”が付きますが，忘れやすいので気を付けてください．ポートが複数ある場合には，間に“,”を挟んで区切ります．

子回路のポートは，すべてこのインスタンス名の右のかっこの中に書く必要があります．また，Verilog HDLでは大文字と小文字は区別されます．子回路のモジュールで大文字で宣言されたポートは，インスタンス宣言でも大文字で記述する必要があります．

Verilog HDLにはポート名を記述しない順番による接続という方法もありますが，順番による接続だとインスタンス宣言を見ただけでは正しく接続されているか分かりづらく，間違いを犯しやすいので，**リスト3(a)**のような名前による接続をお勧めします．

リスト3(b)は，and_combをインスタンス宣言した記述です．

ここではインスタンス名もand_combとしています．インスタンス名は基本的に何でもよいのですが，インスタンス名を見ただけでそれが何のモジュールか分かるようにしておくと，インスタンス宣言を見返す必要がないので便利です．親回路の中に子回路が一つだけのときは，インスタンス名はモジュール名と同じ名前を付け，二つ以上あるときは，and_comb0，and_comb1などのインスタンス名を“モジュール名＋番号”にするとよいでしょう．**リスト3(b)**ではポートに接続される信号がまだ書かれていません．これは以降のステップで記述します．

リスト3　Verilog HDLによるインスタンス宣言の書式と記述例

```
モジュール名 インスタンス名 (. ポート名1 (信号名1),
                              . ポート名2 (信号名2),
                              ・・・) ;
```

(a) 書式

```
and_comb and_comb(.A(     ), .B(     ), .Y(     ));
```

(b) 記述例

VHDL

(1) コンポーネント宣言

リスト4(a)はVHDLのコンポーネント宣言です．

VHDLでは回路（仮に親回路とする）の中に別の回路（仮に子回路とする）を置く場合，接続を記述する前に，まず親回路のアーキテクチャの宣言部分で，子回路をコンポーネント宣言する必要があります．

コンポーネント宣言では，コンポーネント名は子回路のエンティティ名と同一です．また，子回路のすべてのポート名をportに続くかっこの中に記述する必要があります．ポートの記述にはポートの向き（入出力の方向）とデータ型を合わせて記述する必要があり，ポートの向きとデータ型は子回路の中でのエンティティ宣言と合わせる必要があります．

同じ方向，同じデータ型のポートは，“ : ”の左に“ , ”で区切って書き並べることができます．エンティティ宣言と非常に似通っているため，子回路のエンティティ宣言をコピーしてentityをcomponentに置き換えて作ることも多いのですが，エンティティ宣言に存在した“ is ”はなくなっているので，注意が必要です．

リスト4（b）はand_combをコンポーネント宣言した記述例です．

ここでは，ポートA，B，Yともにデータ型がstd_logicになっています．データ型std_logicを使用するためには，ライブラリIEEEのパッケージstd_logic_1164を呼び出す必要があります．

（2）ライブラリ宣言とパッケージ呼び出し

リスト5（a）は，VHDLのライブラリ宣言とパッケージの呼び出しの書式です．

ライブラリ宣言とパッケージ呼び出しはエンティティやアーキテクチャ宣言の外，ファイルの先頭に書きます．2行目の最後のallは，指定したパッケージのすべての宣言を呼び出すことを意味します．ひとつの宣言だけを呼び出す場合には，allの位置にその宣言名を書きます．

ライブラリIEEEの宣言とパッケージstd_logic_1164の呼び出しは，**リスト5（b）**のように記述します．

（3）子回路の接続

VHDLでは，子回路の接続はアーキテクチャ宣言の機能の部分に記述します．**リスト6（a）**はVHDLの子回路を接続するための書式です．

インスタンス名は，親回路の中などで子回路を指定するときの呼び名のようなものです．port mapに続くかっこ中にはポートと信号の接続が記述されます．

“ ポート名1　=>　信号名1 ”は，子回路のポート1と親回路の信号1が接続されていることを記述しています．

リスト6（b）はand_combを接続する記述例です．

インスタンス名は基本的になんでもよいのですが，アーキテクチャ宣言の中で宣言したコンポーネント名をインスタンス名として使うことはできません．ここでは，インスタンス名をuand_combとしています．VHDLでは英字の大文字と小文字を区別することはないので，子回路のエンティティで宣言したポートの大文字と小文字が違っていても，VHDLだけでコンパイルするときには問題となることはありません．しかし，Verilog HDLとの混載シミュレー

リスト4　VHDLによるコンポーネント宣言の書式と記述例

```
component コンポーネント名
   port （ポート名1，ポート名2：ポートの向き　データ型；
          ポート名3           ：ポートの向き　データ型）；
end component;
```

（a）書式

```
component and_comb
   port (A,B  : in  std_logic;
         Y    : out std_logic);
end component;
```

（b）記述例

リスト5　VHDLによるライブラリ宣言とパッケージの呼び出しの書式と記述例

```
library ライブラリ名；
use ライブラリ名．パッケージ名.all;
```

（a）書式

```
library IEEE;
use IEEE.std_logic_1164.all;
```

（b）記述例

リスト6　VHDLによる子回路を接続するための書式と記述例

```
インスタンス名：コンポーネント名
 port map（ポート名1　=>　信号名1，
           ポート名2　=>　信号名2，
           ポート名3　=>　信号名3）;
```

（a）書式

```
uand_comb : and_comb
port map (    A =>        ,
              B =>        ,
              Y =>        );
```

（b）記述例

ションを行う際には問題となるので，子回路のエンティティ名とポートの大文字，小文字はそろえた方が無難です．**リスト6(b)**では，“ => ”の右側には信号名が記述されていませんが，それは以降のステップで記述します．

● 箱の中で入力波形を作る

Verilog HDL

(1)信号の宣言

リスト7(a)は，Verilog HDLにおける信号宣言の書式です．

基本的に箱の中で使用する信号は，このようにデータ型と名前を宣言しなければなりません．同じデータ型の信号は“ , ”で区切って，同じ行に書くことができます．

リスト7(b)は，1ビットの信号SA，SBを宣言した記述例です．

テストベンチでテスト入力として使う信号のデータ型は，1ビットであれば`reg`とします．反対に検証対象の出力ポートに接続するテストベンチで値を与えない1ビットの信号は，`wire`とします．`reg`と`wire`については，またあとで解説します．この宣言は，テスト入力やインスタンス宣言よりも上に書きます．この信号を使っている記述よりも下で宣言すると，コンパイル時にエラーとなります．

(2) initial文

リスト8(a)は，Verilog HDLの`initial`文の書式です．

`initial`文は，シミュレーション開始時点（シミュレーション時間0）で1度だけ“ 式 ”が実行されます．

一つの`initial`文で複数の式を記述したい場合は，**リスト8(b)**のように，`begin`と`end`で囲まれるブロックを作り，`begin`と`end`の間に記述します．

リスト8(a)の式や**リスト8(b)**の`begin`から`end`までの文を，ステートメントと言います．一つのステートメントは，“ ; ”で終わる一つの式か，`begin`～`end`に囲まれた複数の式になります．

リスト7　Verilog HDLによる信号宣言の書式と記述例

```
データ型　信号名1，信号名2
```

(a) 書式

```
reg   SA,SB;
```

(b) 記述例

リスト8(b)には“ #遅延値 ”があります．これはシミュレーションを開始後，0単位時間で式1を実行し，その後100単位時間経過してから式2を実行するという意味になります．単位時間とは“ # ”に続く数値の単位で，設定によってnsにもpsにもできます．

単位時間がnsであれば0nsで式1を実行し，100nsで式2を実行することになります．単位時間のデフォルト値は，たいていのシミュレータでnsになっています．入門レベルのテストベンチでは，単位時間を気にする必要はあまりありません．以降は単位時間をnsとして解説します．

リスト8(c)は，Verilog HDLで**図3**の信号SA，SBの波形を実現したものです．

このコードでは，シミュレーション時間0nsで信号SA，SBに0を代入し，100nsでSAに1をSBに0を代入，200nsでSAに0，SBに1を代入しています．

Verilog HDLの一つの文は，一つの式と“ ; ”でできています．正確には，シミュレーション開始後0nsで信号SAに0を代入し，それから0ns後に信号SBに0を代入し，それから100ns後に信号SAに1を代入します．さらに，それから0ns後に信号SBに0を代入し，さらに100ns後に信号SAに0を代入していることになります．`begin`～`end`の間に書かれた遅延値は，累積して後の式の実行に影響します．

一番最後に書かれた`$finish()`は，そこでシミュレーションが終わるというシステム・タスクです．この行まで実行されるとシミュレーションが終了します．このシステム・タスクがないと，一度実行されたシミュレーションは

リスト8　Verilog HDLによるinitial文の書式と記述例

```
initial 式;
```

(a) 書式1

```
initial begin
        式1;
#遅延値  式2;
end
```

(b) 書式2

```
initial begin
     SA = 0; SB = 0;
#100 SA = 1; SB = 0;
#100 SA = 0; SB = 1;
#100 SA = 1; SB = 1;
#100 $finish();
end
```

(c) 記述例

いつまで経っても終わらなくなってしまいます．

システム・タスクは“$”で始まる関数は$finish()以外にもいろいろありますが，今回はこれだけ覚えればだいじょうぶです．

VHDL

(1)信号の宣言

リスト9(a)は，VHDLにおける信号宣言の書式です．

VHDLでは箱の中で使用するすべて信号は，アーキテクチャ宣言の宣言部分(“is”と“begin”の間)に書かなければいけません．信号宣言にはデータ型が必要で，同じデータ型の信号であれば，“,”で区切って並べて書くことができます．

リスト9(b)は1ビットの信号SA，SB，SYを宣言する記述例です．

1ビットの信号のデータ・タイプは，std_logicとなります．Verilog HDLと違ってVHDLではテストベンチで値を代入する信号も，検証対象の出力ポートに接続する信号も同じデータ型で宣言することができます．

(2)信号への値の代入

リスト10(a)は，VHDLで信号に値を代入する書式です．

センシティビティ・リストの位置には一つの信号名か，“,”で区切られた複数の信号名を書きます．

beginとend processの間の式は，センシティビティ・リストの位置に書かれた信号のどれかが変化したタイミングで実行されます．一番最後の式まで実行されると，次の信号の変化を待って，再び一番上から一番下まで実行されます．センシティビティ・リストを書かないときはかっこも不要で，この場合シミュレーション開始とともに実行を開始し，一番下まで達すると再び一番上から実行を開始し，永久に回り続けます．

式1はシミュレーション時間0で実行されますが，式2はwait forに続く遅延時間が経過してから実行されます．

遅延時間は，時間の数値と時間の単位で書かれます．VHDLでは，一つの文は式と“;”になるので，正確にはシミュレーション時間0で式1が実行され，そのあと0時間後にwaitの文が実行されます．waitの文で遅延時間として設定された時間の経過後，さらに0時間後に式2が実行されます．

リスト10(b)は，VHDLで**図3**の入力信号SA，SBの波形を実現する記述例です．

waitによる遅延は，process文の式の間で累積されます．このコードでは，シミュレーション時間0nsで信号SA，SBに0を代入し，100nsでSAに1をSBに0を代入，200nsでSAに0，SBに1を代入しています．

一番最後に書かれているassert falseは，シミュレーションを終わらせる文になっています．実はVHDLには，シミュレーションを停止するためだけの文というものが存在しません．本来assertというのは条件を設定し，その条件に違反するとメッセージを表示する機能です．そして，条件にERROR(禁止事項)やWARNING(重要注意事項)などのランクをつけて，ランクによっては条件に違反した時点で，シミュレーションを止めてしまうことができます．ここでは条件をfalse(偽)，つまり常に条件は満たされていないと書かれています．条件のランクは，特に記述しないとERRORとなります．通常の設定では，ランクがERRORの条件に違反するとシミュレーションが停止するようになっているので，**リスト10(b)**のassertの文でシミュレーションを止めることができます．ただし，このシミュレーション停止をどのランクにするかは，シミュレーションの環境の設定で変えることができます．もし，シ

リスト9 VHDLによる信号宣言の書式と記述例

```
signal 信号名1，信号名2 ： データ型;
```

(a) 書式

```
signal SA,SB,SY : std_logic;
```

(b) 記述例

リスト10 VHDLによる信号に値を代入する書式と記述例

```
process(センシティビティ・リスト) begin
                式1;
wait for 遅延時間;式2;
end process;
```

(a) 書式

```
process begin
                        SA <= '0'; SB <= '0';
  wait for 100 ns; SA <= '1'; SB <= '0';
  wait for 100 ns; SA <= '0'; SB <= '1';
  wait for 100 ns; SA <= '1'; SB <= '1';
  wait for 100 ns; assert false;
end process;
```

(b) 記述例

4

リスト11　Verilog HDLによるwireの記述例

```
wire  SY;
```

ミュレータの設定がERRORでシミュレーションを停止しないようになっていると，シミュレーションは停止しないので注意してください．シミュレーションが停止しないと，process文の中を永久にループし，テスト入力の値も永久に変化していくことになります．

● 信号を検証対象の回路のポートにつなげる

最後に，テストベンチの信号を検証対象の回路のポートに接続します．VHDLの解説では，入力・出力すべての信号をすでに宣言していましたが，Verilog HDLの解説では，出力ポート側の信号の宣言をしていませんでした．まずは，その解説をします．

Verilog HDL

リスト11は，wire型の1ビットの信号SYを宣言したものです．

書式としては，**リスト7(a)**と同じです．regとwireの使い分けですが，これはVerilog HDLを始めると誰もが最初に戸惑うところです．

まず，おおまかなイメージですが，regは値を保持することができるレジスタ(フリップフロップやラッチ)のイメージです．wireはただの信号線であって，値自体は接続されているフリップフロップや回路の出力に依存して変わるということになります(イメージという言い方で，声のトーンが落ちているのは，always文で組み合わせ回路を作るときなどにおいて，regで宣言した信号がレジスタにならないこのともあるため)．

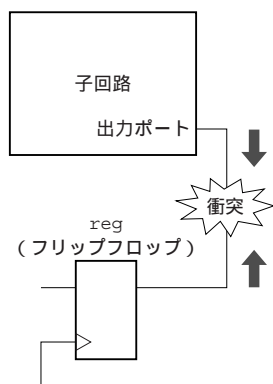



図5　reg型信号と出力ポート
子回路の出力ポートの値は，接続する信号は必ずwire型にする．reg型を接続してしまうと，コンパイル・エラーになる．

リスト12　Verilog HDLによるand_combのテストベンチand_comb_tb

```
module and_comb_tb();
reg   SA,SB;
wire  SY;

and_comb and_comb(.A(SA), .B(SB), .Y(SY));

initial begin
     SA = 0; SB = 0;
#100 SA = 1; SB = 0;
#100 SA = 0; SB = 1;
#100 SA = 1; SB = 1;
#100 $finish();
end

endmodule
```

文法的には，reg型の信号はalways文やinitial文の中で値を代入することができ，一度値が代入されると次に別の値が代入されるまで，その値は保持されます．

reg型で宣言された信号Rに対して，シミュレーション時間0nsで0が代入され，次に100nsで1が代入されたとします．0nsから100nsの間はいつ信号Rを参照しても，値として0を読み出すことができます．

これに対して，wire型で宣言された信号は，ほかの信号に接続されたり子回路の出力ポートに接続され，信号の値はシミュレーション中は常に接続先の信号の値に依存します．wire型で宣言された信号Wが，別の信号Rに接続されているとします．信号Rの値がシミュレーション時間0nsで0になり，100nsで1に変わった場合，シミュレーション時間0nsから100nsの間に信号Wを参照すると，値として0を読み出すことができます．

子回路の出力ポートの値は，子回路の中から出力されるので，接続する信号は必ずwire型になります．reg型を接続してしまうと，コンパイル・エラーを起こします．イメージとしては，レジスタの出力と回路の出力が衝突してしまうと考えてください(**図5**)．

Verilog HDL　VHDL

リスト12(Verilog HDL)と**リスト13**(VHDL)は完成したテストベンチです．検証対象回路のポートにはテストベンチの信号が接続されています．

リスト13 VHDLによるand_combのテストベンチand_comb_tb

```
library IEEE;
use IEEE.std_logic_1164.all;

entity and_comb_tb is
end and_comb_tb;

architecture SIM of and_comb_tb is

component and_comb
   port (A,B  : in  std_logic;
         Y    : out std_logic);
end component;

signal SA,SB,SY : std_logic;

begin

uand_comb : and_comb port map (
    A => SA,
    B => SB,
    Y => SY);

 process begin
                        SA <= '0'; SB <= '0';
   wait for 100 ns; SA <= '1'; SB <= '0';
   wait for 100 ns; SA <= '0'; SB <= '1';
   wait for 100 ns; SA <= '1'; SB <= '1';
   wait for 100 ns; assert false;
 end process;

 end SIM;

 configuration cfg_and_comb_tb of and_comb_tb is
   for SIM
   end for;
 end cfg_and_comb_tb;
```

4 シミュレータによる検証の実施

テストベンチが完成したら，いよいよシミュレーションを行ってください[編集部注1]．信号SYの値を観測し，**図3**のような波形になっていれば，回路and_combは仕様どおりの動作をしていることになり，検証に合格したと言えます．信号SYの波形が**図3**のようになっていなければ，回路and_combは仕様通りになっていないことになり，コードの中を見直して修正しなければなりません．修正の作業は，信号SYの波形が**図3**と一致するまでやらなければいけません．

編集部注1：無償で利用できるシミュレータを，本誌2007年3月号の付属DVD-ROMに収録している．

＊　＊　＊

テストベンチの記述についてさらに詳細な解説を次号以降で連載する予定です．ご期待ください．

やすおか・たかし
(株)エッチ・ディー・ラボ

<筆者プロフィール>
安岡 貴志．東京理科大学 理工学部 数学科卒業．前職のデザインセンターでは，3年間Verilog HDLによるASIC開発に携わる．2002年にエッチ・ディー・ラボに入社し，Verilog HDL，VHDL，SystemCによる開発に従事するほか，同社のトレーニング講師を務める．